UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/692,004	10/24/2003	Alexander J. Kolmykov-Zotov	MSFT-6146/304450.01	8547

41505 7590 07/17/2009

WOODCOCK WASHBURN LLP (MICROSOFT CORPORATION)
CIRA CENTRE, 12TH FLOOR
2929 ARCH STREET
PHILADELPHIA, PA 19104-2891

EXAMINER	
JOSEPH, DENNIS P	

ART UNIT	PAPER NUMBER
2629	

MAIL DATE	DELIVERY MODE
07/17/2009	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/692,004	KOLMYKOV-ZOTOV ET AL.	
	Examiner	Art Unit	
	DENNIS P. JOSEPH	2629	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *11 June 2009*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-20* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-20* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *10/24/2003* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. This Office Action is responsive to amendments filed in application No. 10/692,004 on June 11, 2009. Claims 1-20 are pending and have been examined.

Claim Rejections – 35 USC § 103

2. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

The factual inquiries set forth in *Graham* **v.** *John Deere Co.*, 383 U.S. 1, 148 USPQ 459 (1966), that are applied for establishing a background for determining obviousness under 35 U.S.C. 103(a) are summarized as follows:

1. Determining the scope and contents of the prior art.
2. Ascertaining the differences between the prior art and the claims at issue.
3. Resolving the level of ordinary skill in the pertinent art.
4. Considering objective evidence present in the application indicating obviousness or nonobviousness.

3. **Claims 1-20** rejected under 35 U.S.C. 103(a) as being unpatentable over Applicant's admitted prior art (AAPA), further in view of Price et al. (US 2002/0120607 A1) and Vermeire et al. (US 2001/0025372 A1)

AAPA teaches in Claim 1:

A process for transferring pen data between unmanaged and managed code on a computing device, the unmanaged code being code native to and executed directly by a processor of the computing device **(Figure 3 shows the unmanaged and managed sections as the data is converted)**, the managed code being executed in a common language run-time environment of a framework operating on the computing device **(Figure 3, [0009] shows the common language run-time to manage the various types of codes, such as managed and unmanaged code)**, the common language run-time environment of the framework executing the managed code independent of a type of the processor of the computing device **(Figure 3, note the managed code is a separate section of code, apart from the raw, unmanaged code of the stylus input)**, the process comprising the steps of:

receiving pen data in a component on the computing device written in unmanaged code **(Figure 3, [0009] shows the component 302 which may include pen services)**, the pen data being generated by a digitizer of the computing device upon movement of a stylus with respect to a surface of the digitizer, the pen data including at least one location on the digitizer of the stylus **(Figure 3, [0009] shows the pen device drivers 301 which receives information from a digitizer)**;

the pen input component transferring information related to said pen data to a memory on the computing device designated to be shared between unmanaged code and managed code, **(Figure 3, [0011] discloses the use of a memory for storing the pen information and then converting that data)**;

the pen input component transferring said information in said shared memory to a stylus input subsystem of an application on the computing device, the stylus input subsystem being

separate from the pen input component and being written in managed code and executed in the common language run-time environment **(Figure 3, [0009] shows the application which receives the managed code after it has been converted from the unmanaged code in the CLR. This is done in the runtime callable wrapper 305 which contains a type library importer. [0009] discloses 305 and 306 import the data from the pen input component (read these as being a stylus input subsystem, which is reasonable given a subsystem is just a follow-up to the primary pen component). [0009] discloses command that can be invoked to retrieve information from pen component 302 and the callable wrapper can obviously, if not inherently, perform other actions to convert the unmanaged code)**;

the stylus input subsystem receiving the pen input component **(Figure 3, [0009] show the pen component's information is retrieved by 305 which invokes a command to get the data)**; and

the stylus input subsystem submitting with a retrieval command to the shared memory to retrieve said information from said shared memory, the retrieval command as submitted by the stylus input subsystem comprising a P-invoke method "GetData" which takes into account a context of the stylus input subsystem. **(Please note [0009] and [0076] for example. Both detail marshalling code and converting code using 304, 305 and 306. It is also obvious and well known that GetData is a commonly used command in CLR and Examiner asserts Official Notice to this. In addition, it is obvious, if not inherent, that there is a command to initiate the retrieval of data in Figure 3)**; but

AAPA does not explicitly teach of a "mutual exclusion" shared memory so that non-simultaneous sharing of code can be achieved between the unmanaged and managed sections.

However, the use of mutex is well known in the art.

To emphasize, Price teaches of using a mutex to allow multiple programs to share the same resource, but not simultaneously, (Price, [0060]).

Therefore, it would be obvious to one of ordinary skill in the art at the time of the invention to integrate the mutex, as taught by Price, with the AAPA's device with the motivation that by non-simultaneously sharing the mutex, sequential operation can be done and errors can be prevented from different programs trying to access the same object at the same time.

AAPA also does not explicitly teach of a pointer to reference different parts of the memory, either to store or retrieve them using the pointers. However, pointers are well known for this function, to point to addresses for retrieval.

To emphasize, in the same field of endeavor, framework systems, Vermeire teaches of using memory addresses or pointers in the runtime framework as a means to access data, (Vermeire, [0033]).

Therefore it would be obvious to one of ordinary skill in the art at the time of the invention to integrate the use of pointers, as taught by Vermeire, with the AAPA with the motivation that it is common to use pointers in the art as well as for better efficiency, memory management and to prevent memory leaks.

AAPA, Price and Vermeire teach in Claim 2:

The process according to claim 1, further comprising the steps of:

transferring additional information from said at least in part managed application to said shared memory **([0011] discloses a memory. Price has been combined with AAPA to teach of a mutex as well)**;

transferring a pointer that points to said additional information to said component; retrieving said additional information from said shared memory. **(The combination with Vermeire teaches to use pointers to access and store data in memory)**

AAPA teaches in Claim 4:

The process according to claim 1, further comprising the step of: exchanging information through a COM interface. **(Figure 3, [0009] shows the COM 303)**

AAPA teaches in Claim 5:

The process according to claim 1, said component being a pen services component. **(Figure 3, [0009] discloses the component 302 may include pen services)**

AAPA teaches in Claim 6:

The process according to claim 1, said application including a pen input managed client. **(This is obvious in light of Figure 3's unmanaged and managed code relating to pen input from a digitizer)**

AAPA teaches in Claim 7:

The process according to claim 1, said component receiving input from at least one pen device driver. **(Figure 3, [0009] shows the pen device drivers 301)**

AAPA teaches in Claim 8:

A system for transferring information between unmanaged code and managed code on a computing device, the unmanaged code being code native to and executed directly by a processor of the computing device **(Figure 3 shows the unmanaged and managed sections as the data is converted)**, the managed code being executed in a common language run-time environment of a framework operating on the computing device **(Figure 3, [0009] shows the common language run-time to manage the various types of codes, such as managed and unmanaged code)**, the common language run-time environment of the framework executing the managed code independent of a type of the processor of the computing device **(Figure 3, note the managed code is a separate section of code, apart from the raw, unmanaged code of the stylus input)**, the system comprising:

a shared memory on the computing device designated to be shared between unmanaged code and managed code **(Figure 3, [0011] discloses the use of a memory for storing the pen information and then converting that data)**;

a pen input component on the computing device that is written in unmanaged code and that receives pen data **(Figure 3, [0009] shows the component 302 which may include pen services)**, the pen data being generated by a digitizer of the computing device upon movement of a stylus with respect to a surface of the digitizer, the pen data including at least one location on the digitizer of the stylus **(Figure 3, [0009] shows the pen device drivers 301 which receives information from a digitizer)**, the pen input component transferring information relating to said pen data to said memory and transferring said information in the memory to a stylus input subsystem of an application on the computing device, the stylus input subsystem being separate from the pen input component and being managed code and executed in the common language run-time environment **(Figure 3, [0009] shows the application which receives the managed code after it has been converted from the unmanaged code in the CLR. This is done in the runtime callable wrapper 305 which contains a type library importer. [0009] discloses 305 and 306 import the data from the pen input component (read these as being a stylus input subsystem, which is reasonable given a subsystem is just a follow-up to the primary pen component). [0009] discloses command that can be invoked to retrieve information from pen component 302 and the callable wrapper can obviously, if not inherently, perform other actions to convert the unmanaged code)**;

said stylus input subsystem having managed code receiving the transferred pointer from the pen input component and submitting the receiving pointer with a retrieval command to the

shared memory to retrieve said information from said shared memory by way of the transferred pointer, the retrieval command as submitted by the stylus input subsystem comprising a P-invoke method "GetData", which takes into account a context of the stylus input subsystem **(Please note [0009] and [0076] for example. Both detail marshalling code and converting code using 304, 305 and 306. It is also obvious and well known that GetData is a commonly used command in CLR and Examiner asserts Official Notice to this. In addition, it is obvious, if not inherent, that there is a command to initiate the retrieval of data in Figure 3)**; but

AAPA does not explicitly teach of a "mutual exclusion" shared memory so that non-simultaneous sharing of code can be achieved between the unmanaged and managed sections.

However, the use of mutex is well known in the art.

To emphasize, Price teaches of using a mutex to allow multiple programs to share the same resource, but not simultaneously, (Price, [0060]).

Therefore, it would be obvious to one of ordinary skill in the art at the time of the invention to integrate the mutex, as taught by Price, with the AAPA's device with the motivation that by non-simultaneously sharing the mutex, sequential operation can be done and errors can be prevented from different programs trying to access the same object at the same time.

AAPA also does not explicitly teach of a pointer to reference different parts of the memory, either to store or retrieve them using the pointers. However, pointers are well known for this function, to point to addresses for retrieval.

To emphasize, in the same field of endeavor, framework systems, Vermeire teaches of using memory addresses or pointers in the runtime framework as a means to access data, (Vermeire, [0033]).

Therefore it would be obvious to one of ordinary skill in the art at the time of the invention to integrate the use of pointers, as taught by Vermeire, with the AAPA with the motivation that it is common to use pointers in the art as well as for better efficiency, memory management and to prevent memory leaks.

AAPA teaches in Claim 9:

The system according to claim 8, said component exposing a COM interface. **(Figure 3, [0009] shows the COM 303)**

AAPA teaches in Claim 11:

The system according to claim 8, said component including a pen services component. **(Figure 3, [0009] discloses the component 302 may include pen services)**

AAPA teaches in Claim 12:

The system according to claim 8, further comprising: at least one pen device driver sending information to said component. **(Figure 3, [0009] shows the pen device drivers 301)**

AAPA teaches in Claim 13:

The system according to claim 8, further comprising: said application including a pen input managed client. **(This is obvious in light of Figure 3's unmanaged and managed code relating to pen input from a digitizer)**

AAPA teaches in Claim 14:

A computer-readable storage medium having a program stored thereon for transferring information related to ink between an unmanaged pen input component and a managed stylus input subsystem of an application on a computing device **(Figure 3 shows the unmanaged and managed sections as the data is converted)**, the unmanaged pen input component being native to and executed directly by a processor of the computing device **(Figure 3, [0009] shows the common language run-time to manage the various types of codes, such as managed and unmanaged code)**, the managed stylus input subsystem of the application being separate from the unmanaged pen input component and being executed in a common language run-time environment of a framework operating on the computing device **(Figure 3, [0009] shows the common language run-time to manage the various types of codes, such as managed and unmanaged code)**, the common language run-time environment of the framework operating on the computing device, the common language run-time environment of the framework executing the managed code independent of a type of the processor of the computing device **(Figure 3,**

note the managed code is a separate section of code, apart from the raw, unmanaged code of the stylus input), said program comprising the steps of:

receiving pen data in the unmanaged component on the computing device **(Figure 3, [0009] shows the component 302 which may include pen services)**, the pen data being generated by a digitizer of the computing device upon movement of a stylus with respect to a surface of the digitizer, the pen data including at least one location on the digitizer of the stylus **(Figure 3, [0009] shows the pen device drivers 301 which receives information from a digitizer)**;

the pen input component transferring information related to said pen data to a shared memory on the computing device designated to be shared between unmanaged pen input component and the managed stylus input subsystem **(Figure 3, [0011] discloses the use of a memory for storing the pen information and then converting that data)**;

the stylus input subsystem receiving from the pen input component **(Figure 3, [0009] show the pen component's information is retrieved by 305 which invokes a command to get the data. Figure 3, [0009] shows the application which receives the managed code after it has been converted from the unmanaged code in the CLR. This is done in the runtime callable wrapper 305 which contains a type library importer. [0009] discloses 305 and 306 import the data from the pen input component (read these as being a stylus input subsystem, which is reasonable given a subsystem is just a follow-up to the primary pen component). [0009] discloses command that can be invoked to retrieve information from pen component 302 and the callable wrapper can obviously, if not inherently, perform other actions to convert the unmanaged code)**; and

the stylus input subsystem submitting with a retrieval command to the shared memory to retrieve said information from said shared memory, the retrieval command as submitted by the stylus input subsystem comprising a P-invoke method "GetData" which takes into account a context of the stylus input subsystem **(Please note [0009] and [0076] for example. Both detail marshalling code and converting code using 304, 305 and 306. It is also obvious and well known that GetData is a commonly used command in CLR and Examiner asserts Official Notice to this. In addition, it is obvious, if not inherent, that there is a command to initiate the retrieval of data in Figure 3)**; but

AAPA does not explicitly teach of a "mutual exclusion" shared memory so that non-simultaneous sharing of code can be achieved between the unmanaged and managed sections.

However, the use of mutex is well known in the art.

To emphasize, Price teaches of using a mutex to allow multiple programs to share the same resource, but not simultaneously, (Price, [0060]).

Therefore, it would be obvious to one of ordinary skill in the art at the time of the invention to integrate the mutex, as taught by Price, with the AAPA's device with the motivation that by non-simultaneously sharing the mutex, sequential operation can be done and errors can be prevented from different programs trying to access the same object at the same time.

AAPA also does not explicitly teach of a pointer to reference different parts of the memory, either to store or retrieve them using the pointers. However, pointers are well known for this function, to point to addresses for retrieval.

To emphasize, in the same field of endeavor, framework systems, Vermeire teaches of using memory addresses or pointers in the runtime framework as a means to access data, (Vermeire, [0033]).

Therefore it would be obvious to one of ordinary skill in the art at the time of the invention to integrate the use of pointers, as taught by Vermeire, with the AAPA with the motivation that it is common to use pointers in the art as well as for better efficiency, memory management and to prevent memory leaks.

AAPA, Price and Vermeire teach in Claim 15:

The computer-readable storage medium according to claim 14, said program further comprising the steps of:

transferring additional information from said at least in part managed application to said shared memory **([0011] discloses a memory. Price has been combined with AAPA to teach of a mutex as well)**;

transferring a pointer that points to said additional information to said component; retrieving said additional information from said shared memory. **(The combination with Vermeire teaches to use pointers to access and store data in memory)**

AAPA teaches in Claim 17:

The computer-readable storage medium according to claim 14, said program further comprising the step of: exchanging information through a COM interface. **(Figure 3, [0009] shows the COM 303)**

AAPA teaches in Claim 18:

The computer-readable storage medium according to claim 14, said component being a pen services component. **(Figure 3, [0009] discloses the component 302 may include pen services)**

AAPA teaches in Claim 19:

The computer-readable storage medium according to claim 14, said application including a pen input managed client. **(This is obvious in light of Figure 3's unmanaged and managed code relating to pen input from a digitizer)**

AAPA teaches in Claim 20:

The computer-readable storage medium according to claim 14, said component receiving input from at least one pen device driver. **(Figure 3, [0009] shows the pen device drivers 301)**

As per Claims 3, 10 and 16:

These claims are directed to the use of a P-invoke style interface. Examiner takes Official Notice as to the use of P-invoke interfacing. This is common in the art as a software means and please also see the combination with Vermeire.

Response to Arguments

4. Applicant's arguments considered, but are in moot in view of the new rejection.

Applicant's attorney is thanked for the telephone interview on May 19, 2009 to discuss and case and ways to advance prosecution. As a result of that, a new grounds of rejection has been applied because of the claim amendments.

Applicant heeded the examiner's suggestion to claim the stylus input subsystem. However, the functionality is not really described, or not in a way to overcome the current rejection. To explain, all that is claimed is that the subsystem receives a pointer from the pen input component and issues a command, "GetData", to retrieve data from the memory. More detail is needed in this area as this is not enough to overcome Figure 3 of the APAA and ordinary skill in the art. The term for "GetData" is actually very popular in CLR and is a well known command to programmers. Also, this particular function, of retrieving data from a memory, in this case, the unmanaged and managed code, is respectfully, inherent as the CLR needs some way to access the pen data. The particular claimed command just seems to be obvious in examiner's opinion. Perhaps there are other commands that aren't popular/obvious to overcome the rejection? Examiner also asserted Official Notice to this and would be happy to provide references if Applicant feels this particular command is not known in the art.

Applicant's arguments with regards to pointers and the subsequent amendments were enough to alter the grounds of rejection. Vermeire has now been combined to teach of using pointers for accessing memories in the CLR environment and this is done to support the well known statement that Examiner has asserted earlier.

From looking at Applicant's various figures, there seems to be a few ways to overcome the current rejection such as by claiming the various ways to transmit stylus packets, the compilers with regards to coding, the first and second components. Respectfully, Figures 3 and 4 have been the focus of the prosecution so far, but it doesn't look as though the AAPA teaches of the other figures of the invention. Also, examiner suggests that Applicant perhaps claim the actual Microsoft .NET framework, just a thought.

Applicant is advised to try claiming the above suggestions or better claim the input subsystem or more commands (that aren't inherent or obvious like retrieving data) to overcome the current rejection.

Conclusions

5. Applicant's amendments and non-persuasive arguments necessitated the new ground(s) of rejection presented in this Office action. Accordingly, **THIS ACTION IS MADE FINAL**. See MPEP 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

Any inquiry concerning this communication or earlier communications from the examiner should be directed to DENNIS P. JOSEPH whose telephone number is (571)270-1459. The examiner can normally be reached on Monday-Friday, 8am-5pm.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Amr Awad can be reached on 571-272-7764. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

DJ

/Amr Awad/

Supervisory Patent Examiner, Art Unit 2629